# C++0x むーぶせまんちくす

くらいおらいと

# 自己紹介

- 名前: Cryolite
- ふりがな: くらいおらいと
- 出身地: 大阪市
- 興味のあること: C++, STL, Boost, C++0x
- ブログなど: “Cryolite” で検索
- お仕事:
  - 日本語の文を単語に分けるお仕事
  - 機械に学習させるお仕事

# ムーヴ

C/C++ においては・・・

オブジェクト操作の基本はコピー

汎用ライブラリの運用実績

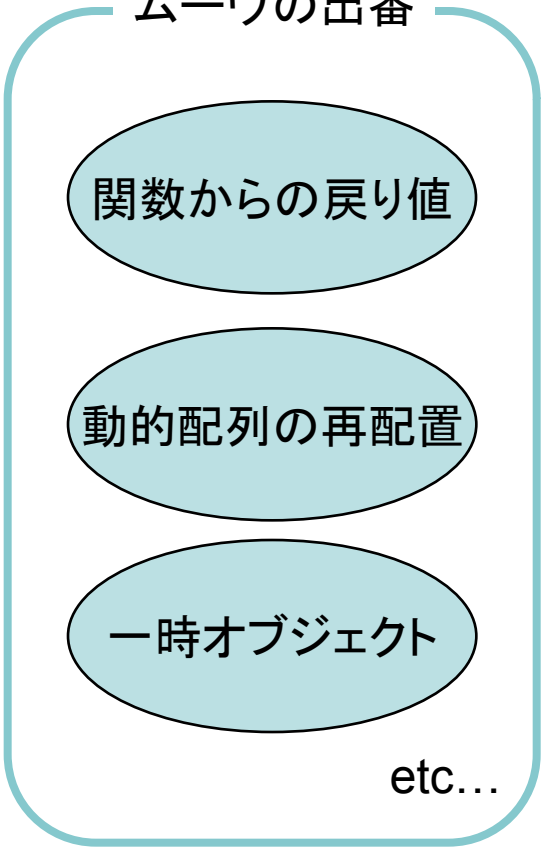
C++では

ムーヴも重要な操作じゃね？

C++03 ではムーブは面倒

C++0x では

ムーヴのための言語サポート！



# セッションの流れ

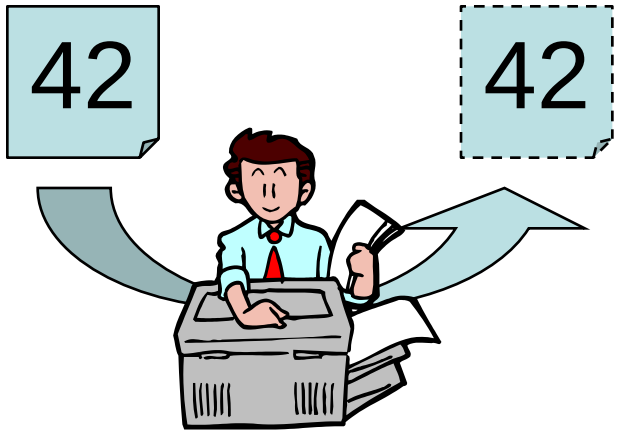
- C++03 におけるコピーの重要性とその限界
- ムーヴの導入 – 動的配列再配置の例から
- ムーヴの活用例
  - ローカルオブジェクトの return
  - 一時オブジェクト
- ムーヴのための C++0x 言語機能 – 右辺値参照
- 明示的なムーヴ
- まとめ

# セッションの流れ

- C++03 におけるコピーの重要性とその限界
- ムーヴの導入 – 動的配列再配置の例から
- ムーヴの活用例
  - ローカルオブジェクトの return
  - 一時オブジェクト
- ムーヴのための C++0x 言語機能 – 右辺値参照
- 明示的なムーヴ
- まとめ

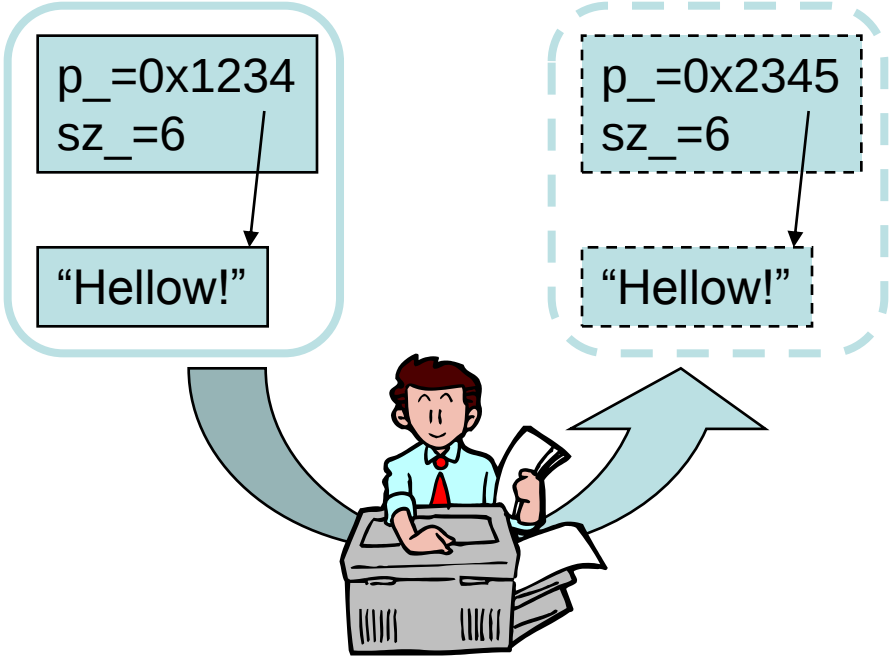
# コピー

```
int i = 42;
int j = i;
```



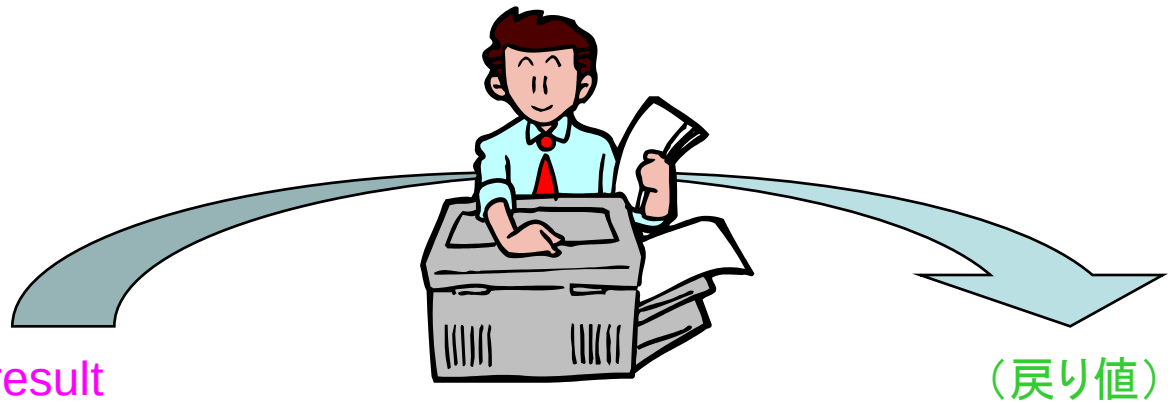

```
class String
{
   .....
private:
   char *p_;
   std::size_t sz_;
};
```

```
String s("Hello!");
String t = s;
```



# コピーの出番 – 戻り値の生成

```
int plus(int lhs, int rhs)
{
  int result = lhs;
  result += rhs;
  return result;
}
```



result

42

（戻り値）

42

# コピーの出番 – 一時オブジェクトからのコピー

```
int i = plus(18, 24);
```

result

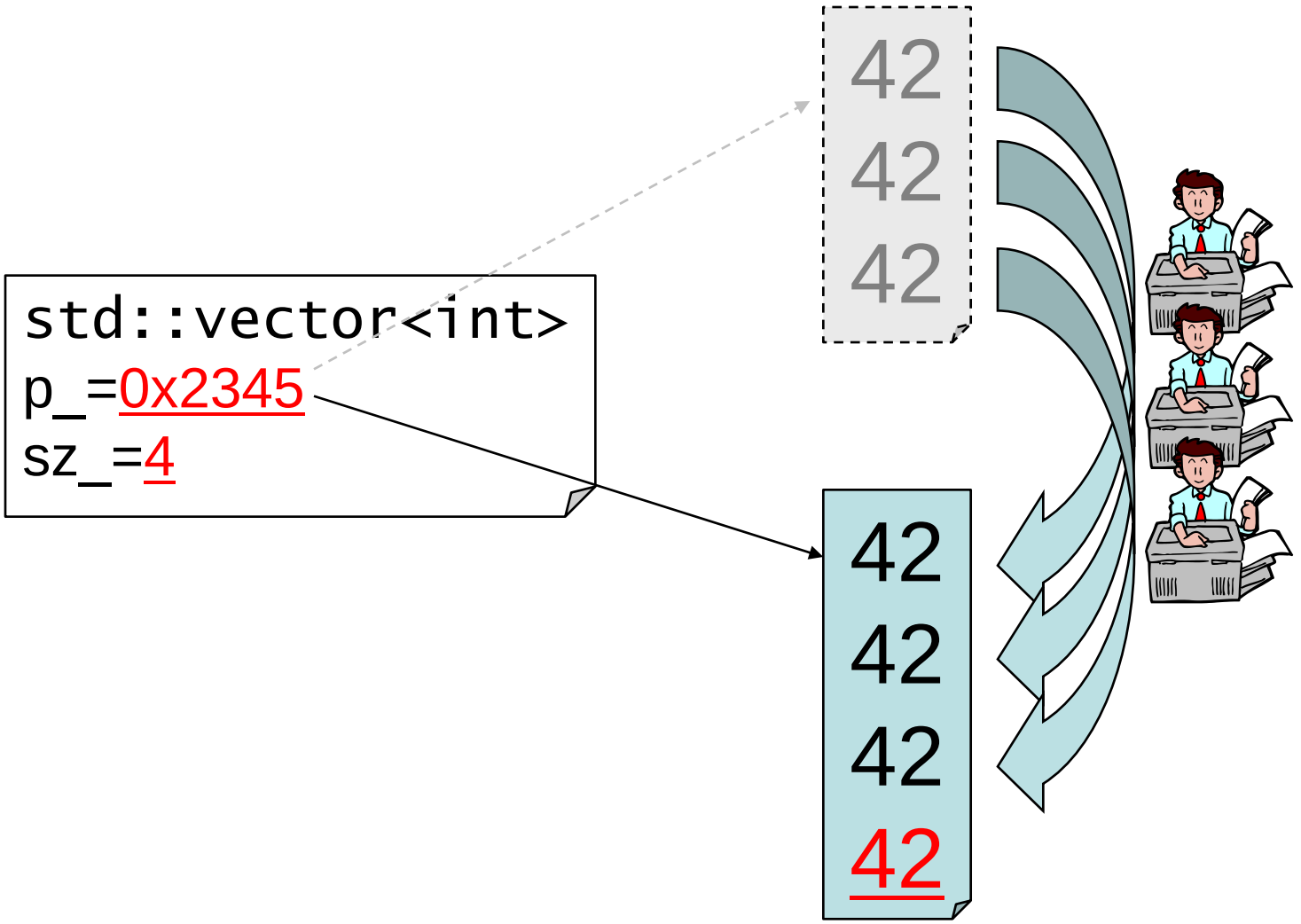
42

（戻り値）

42

# コピーの出番 – 動的配列の再配置

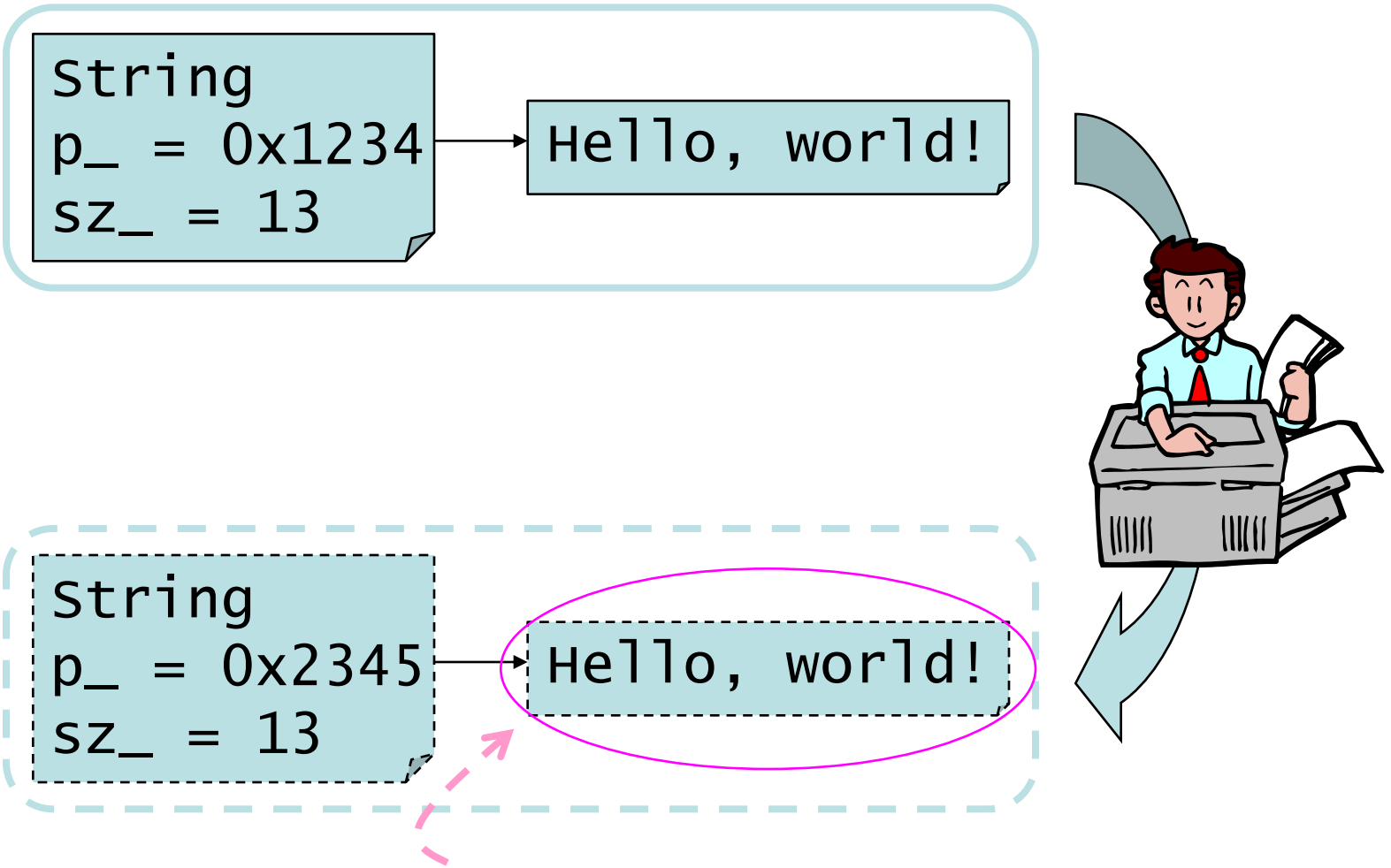
長さ3，capacity=3 の int の動的配列に int オブジェクトを1つ追加



# コピーの限界

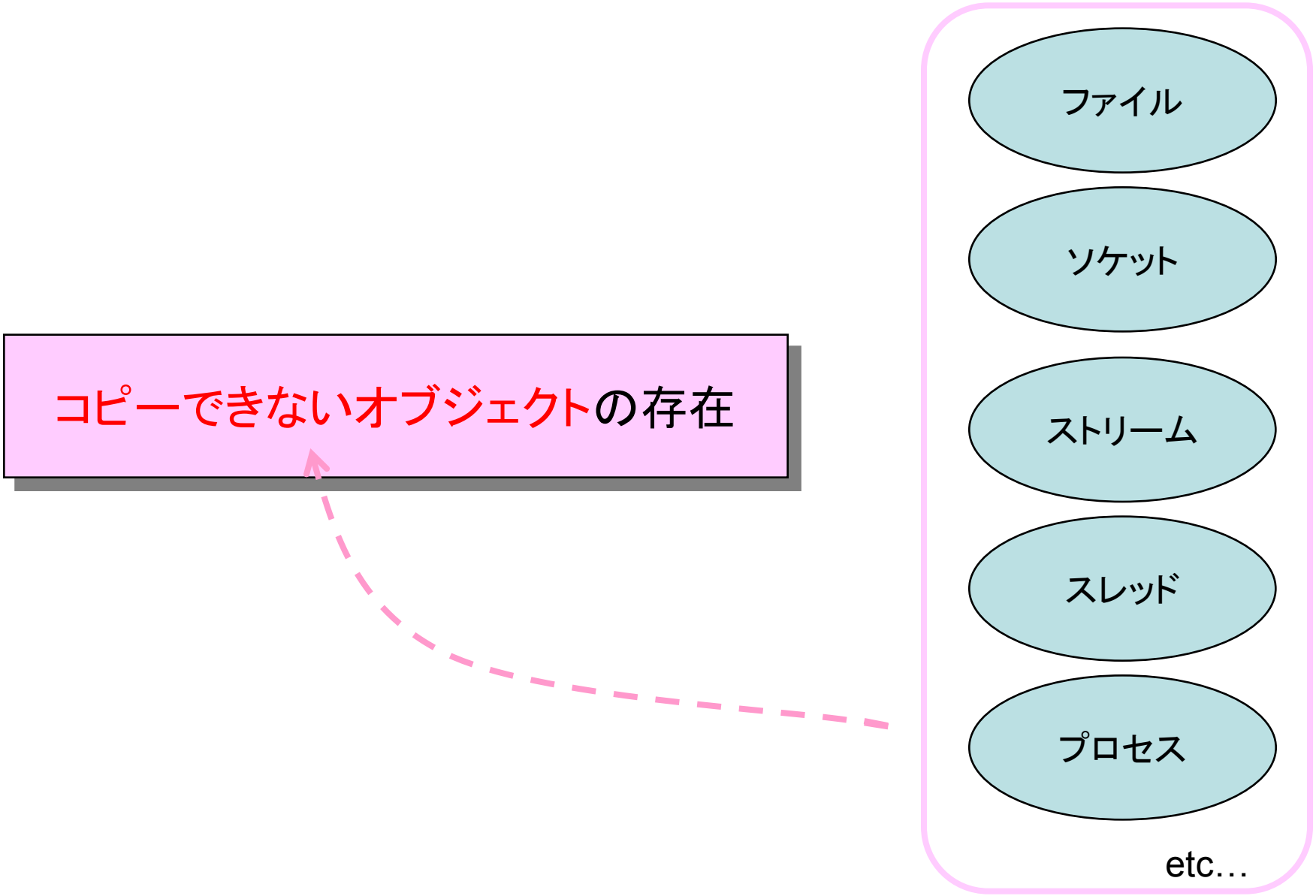
- コピーが重いオブジェクト
- コピーできないオブジェクト

# コピーの弱点その1 – コピーが重いオブジェクトの存在




- フリーストア (ヒープ) 上のメモリ領域が必要
- 可能ならばこの領域の確保を回避したい

# コピーの弱点その2 – コピーできないオブジェクトの存在



# コピーの限界

- コピーの重いオブジェクトを扱うと効率が悪いのか？
  - 重いオブジェクトは以下の点で非効率；
    - 動的配列の再配置
    - 戻り値として指定
    - 一時オブジェクトの生成

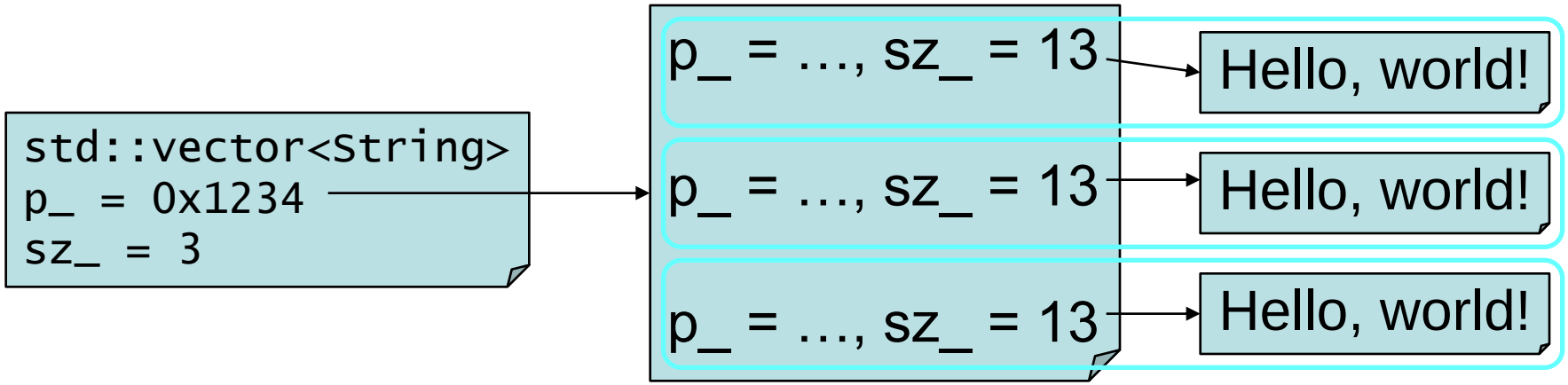
- コピーできないオブジェクトを扱えないのか？
  - コピーできないオブジェクトは以下が直接できない；
    - 動的配列に乗せる
    - 戻り値として指定
    - 一時オブジェクトからコピー

# セッションの流れ

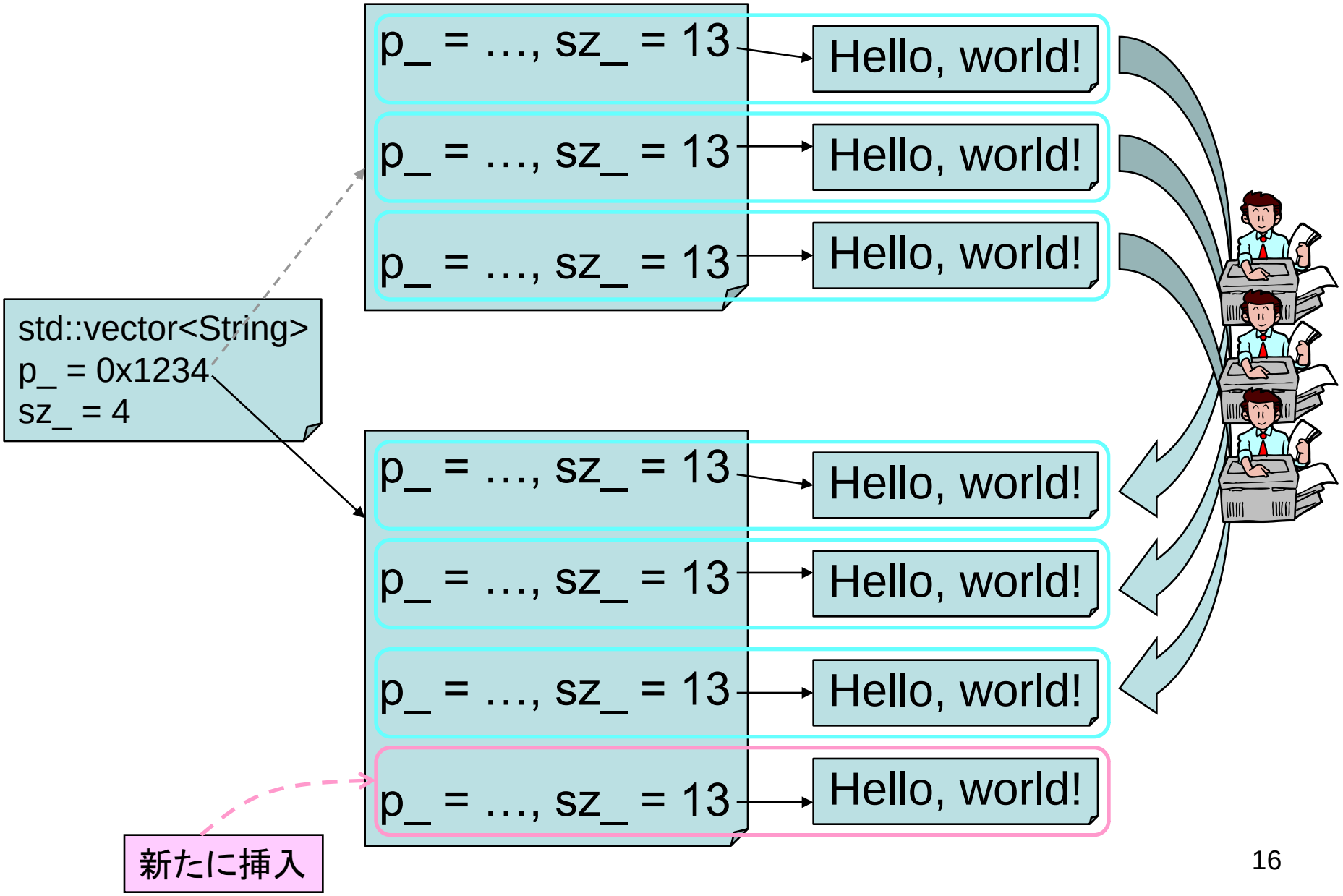
- C++03 におけるコピーの重要性とその限界
- **ムーヴの導入 – 動的配列再配置の例から**
- ムーヴの活用例
  - ローカルオブジェクトの return
  - 一時オブジェクト
- ムーヴのための C++0x 言語機能 – 右辺値参照
- 明示的なムーヴ
- まとめ

# ムーヴの導入 – 動的配列の再配置の例

長さ3, capacity=3 の動的配列 (`std::vector`) の例

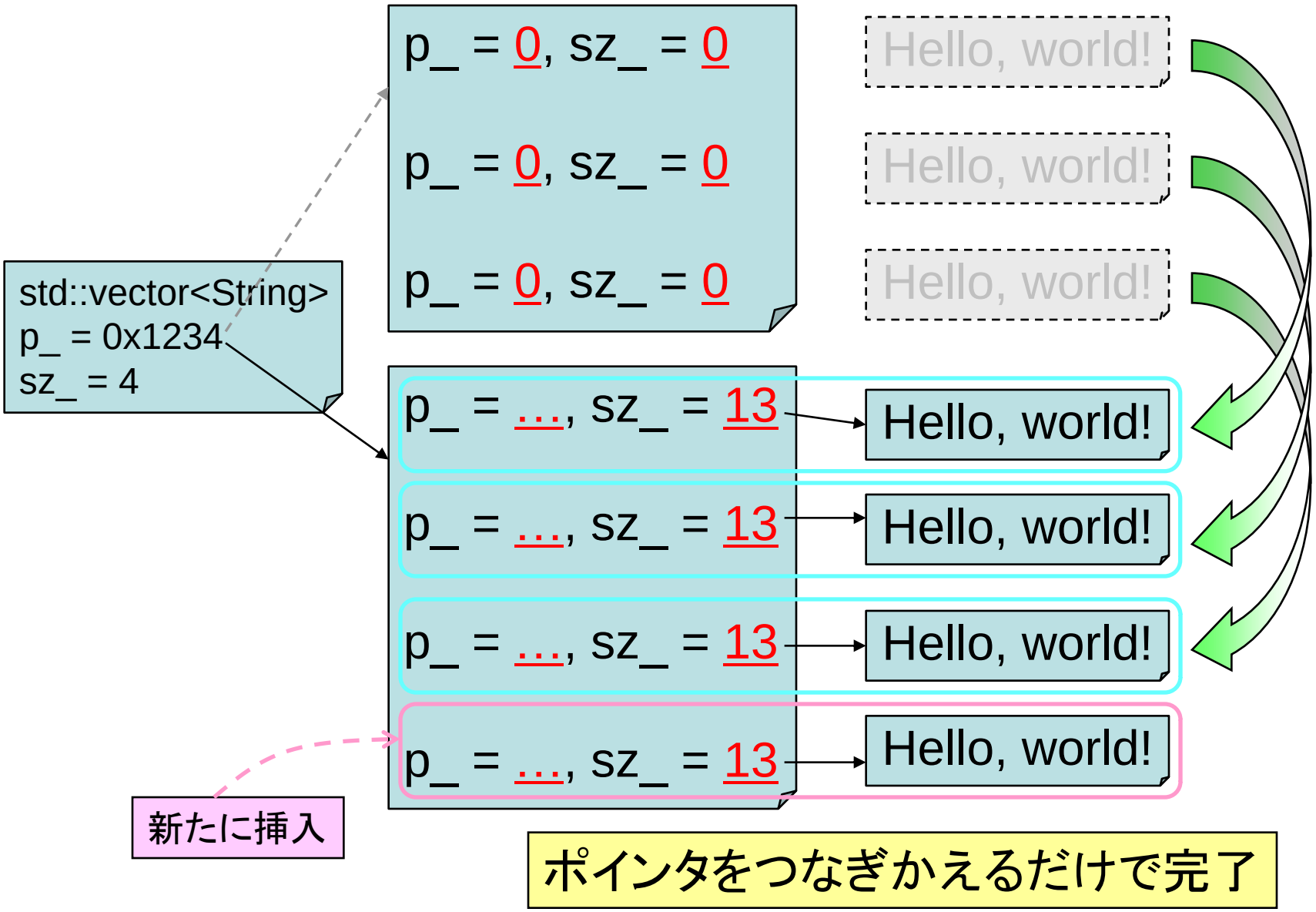


# ムーヴの導入 – 動的配列の再配置の例



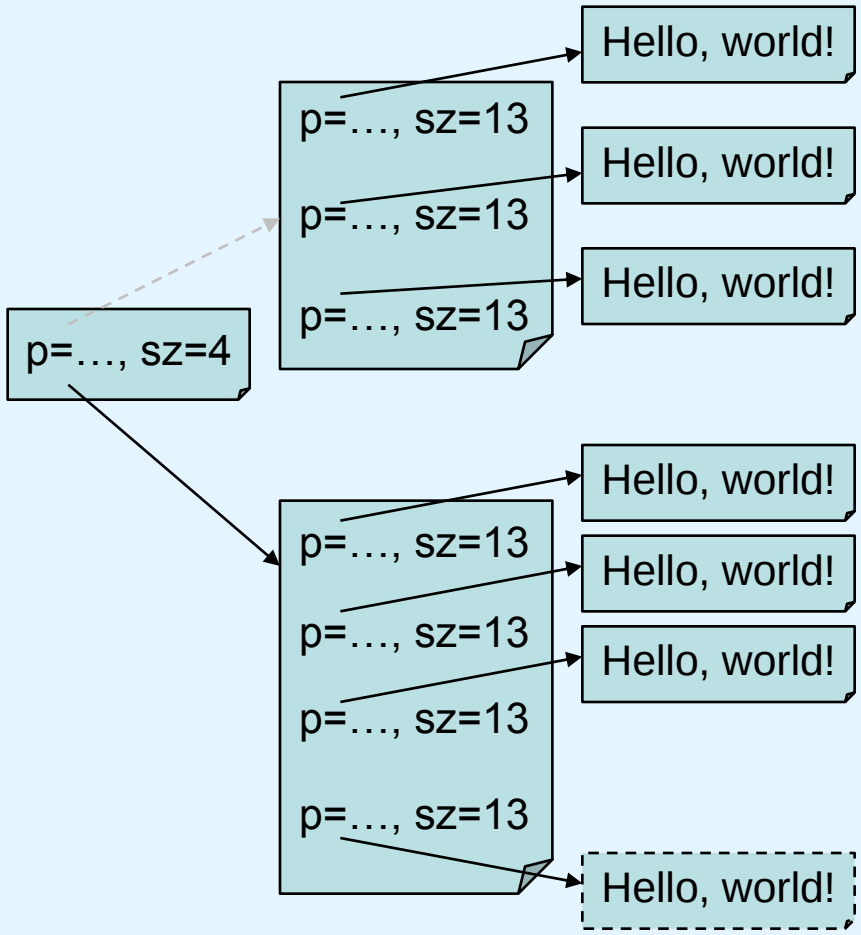
# ムーヴの導入 – 動的配列の再配置の例

もしも，文字列オブジェクトの private メンバを触っても良いとしたら？



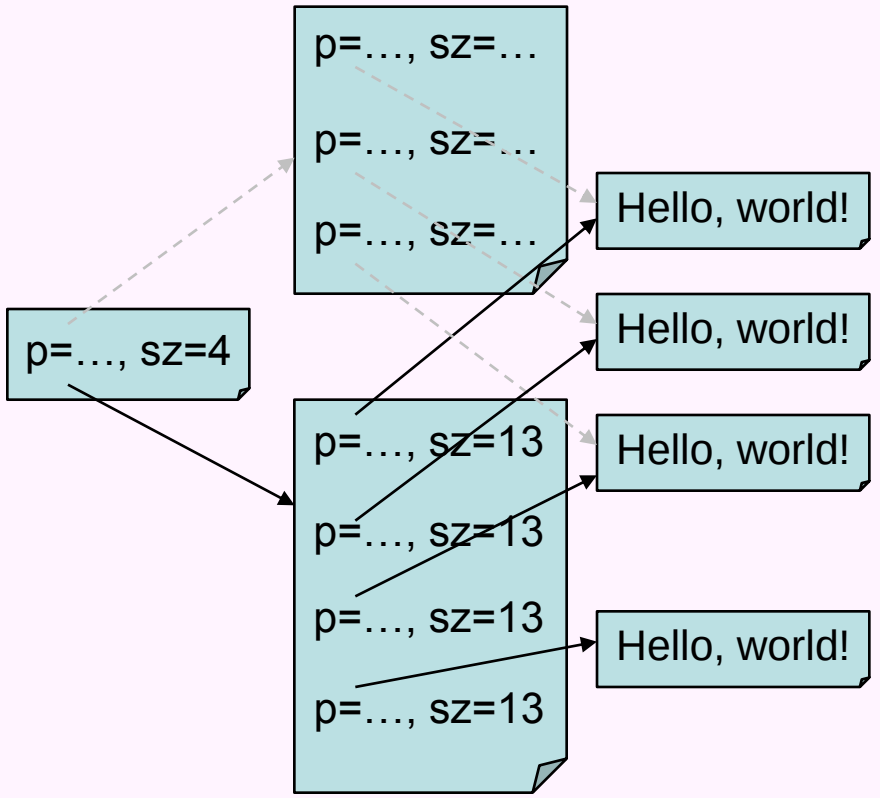

ポインタをつなぎかえるだけで完了

# コピー vs. 直接操作




コピーによる操作 → 〇抽象化

コピーによるリソースの再確保 → ×効率

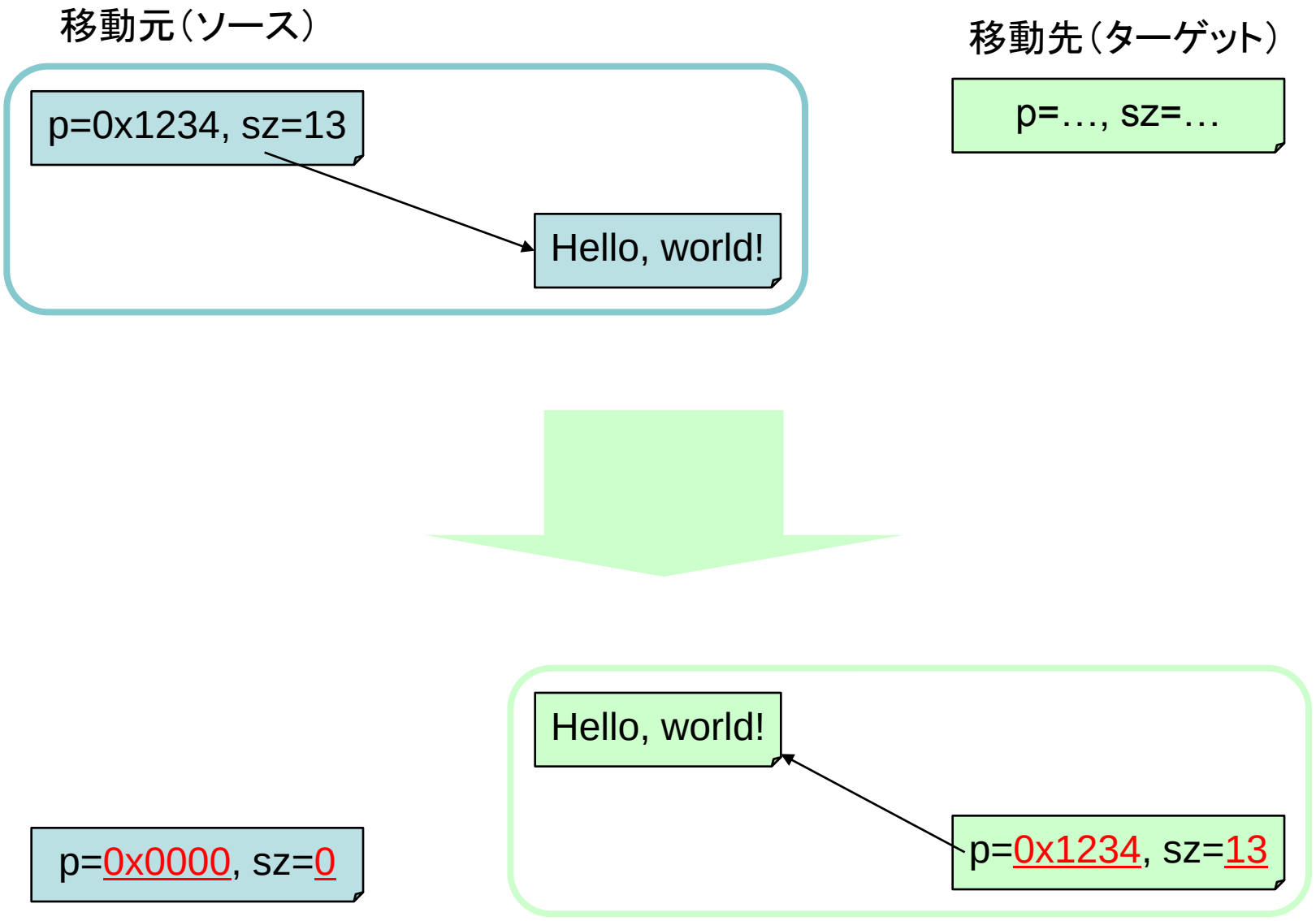



ポインタのつなぎかえのみ → 〇効率

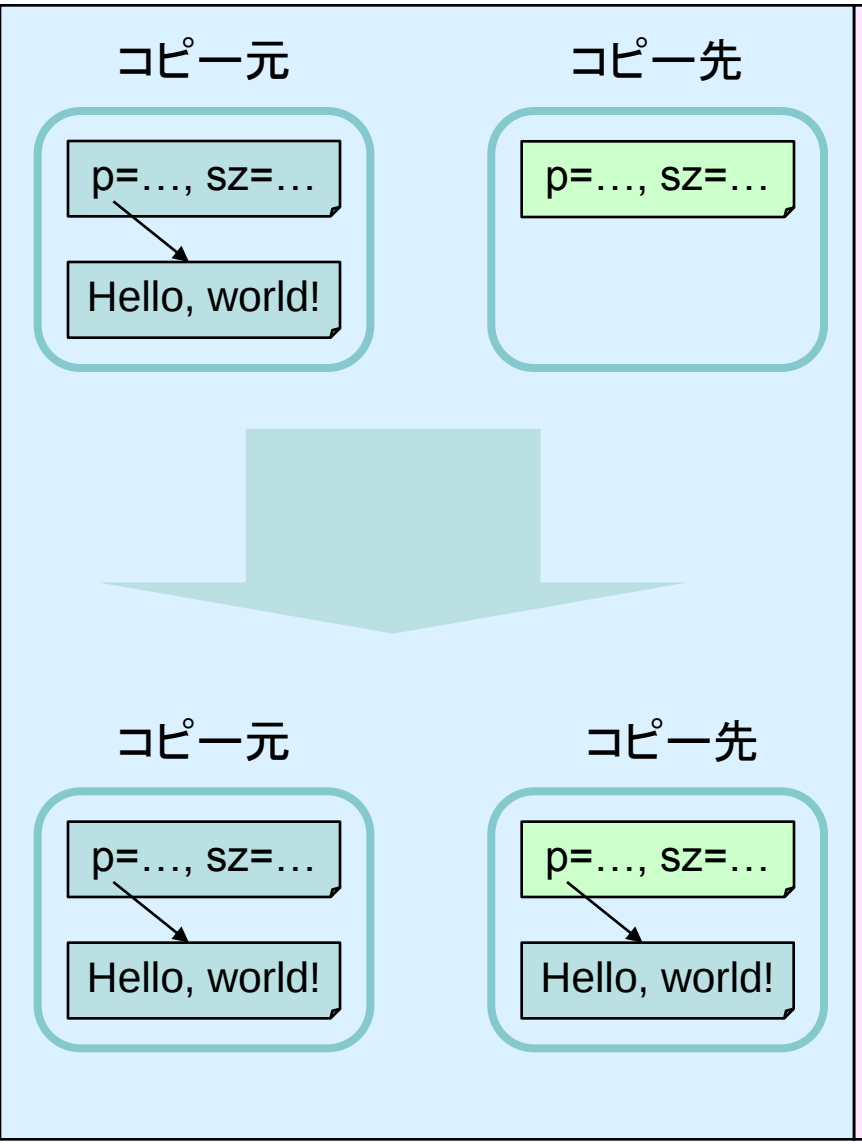
private メンバ直接操作 → ×抽象化

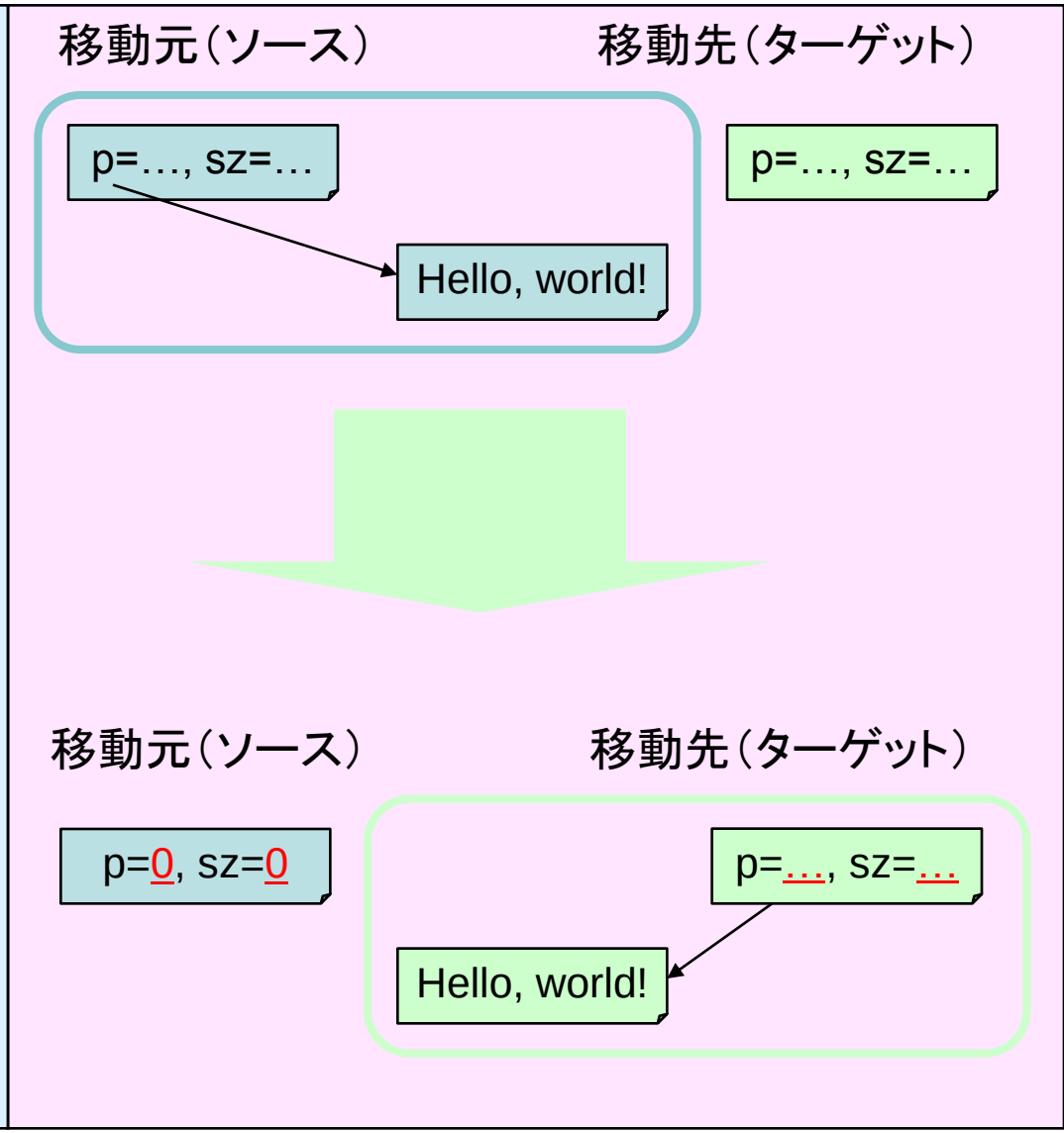
右側における文字列オブジェクトの操作を抽象化

# ムーヴ – 文字列オブジェクトの例



# コピーとムーヴの比較




コピー元は不変

ムーヴ元は変わるかも知れない

# セッションの流れ

- C++03 におけるコピーの重要性とその限界
- ムーヴの導入 – 動的配列再配置の例から
- ムーヴの活用例
  - ローカルオブジェクトの return
  - 一時オブジェクト
- ムーヴのための C++0x 言語機能 – 右辺値参照
- 明示的なムーヴ
- まとめ

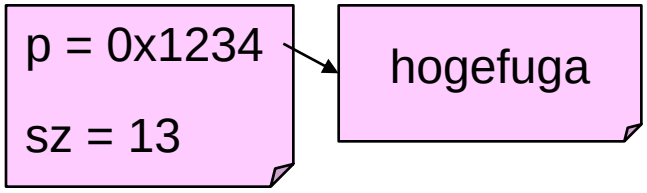
# 戻り値へのコピー

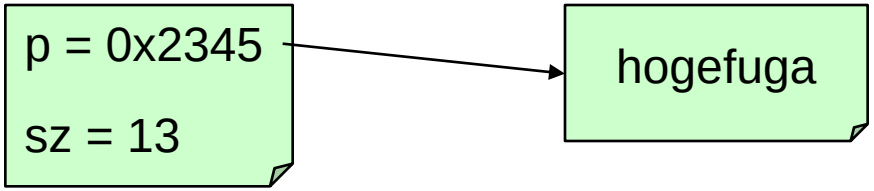
```
String operator+(
  const String& lhs,
  const String& rhs)
{
  String result(lhs);
  // lhs に rhs の文字列を連結
  return result;
}
```

return 文以降 result は使わない

↓

ムーヴで十分！

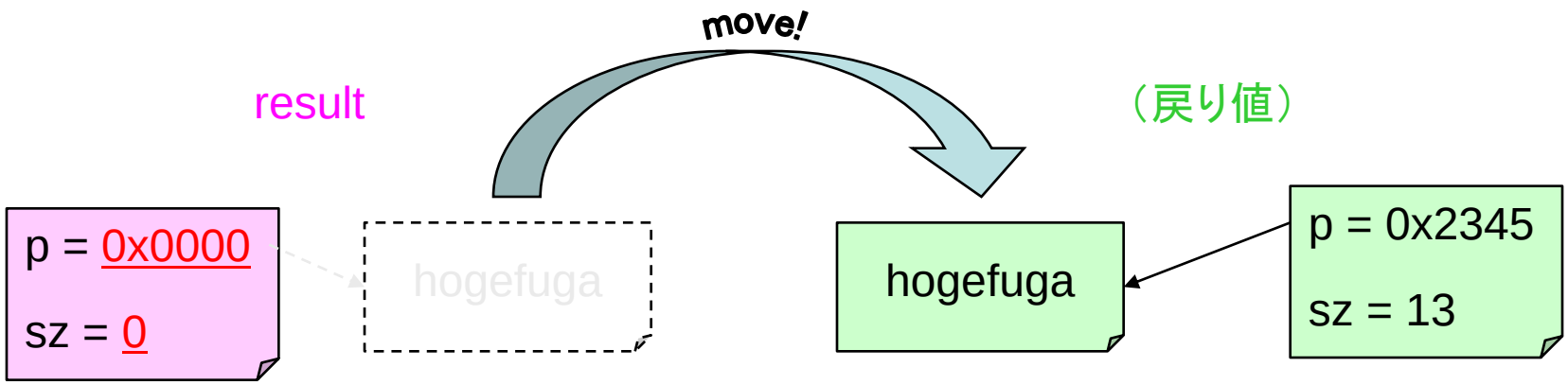
# 戻り値へのムーヴ

```
String operator+(
  const String& lhs,
  const String& rhs)
{
  String result(lhs);
  // lhs に rhs の文字列を連結
  return result;
}
```

return 文以降 result は使わない

↓

ムーヴで十分！

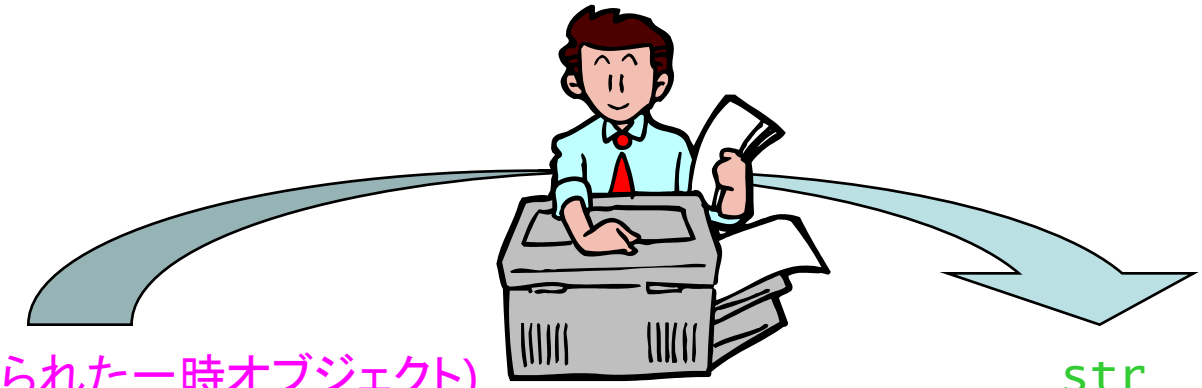


# 一時オブジェクトとコピー

```
String h("Hello, ");
String w("world!");
String str = h + w;
```

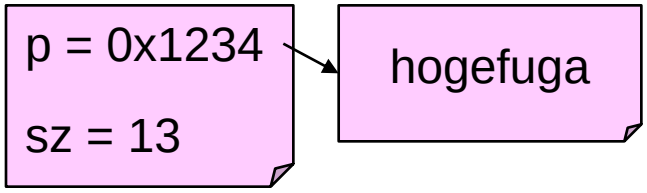
'h + w' で作られる一時オブジェクトは
'str' の初期化以外に使われない

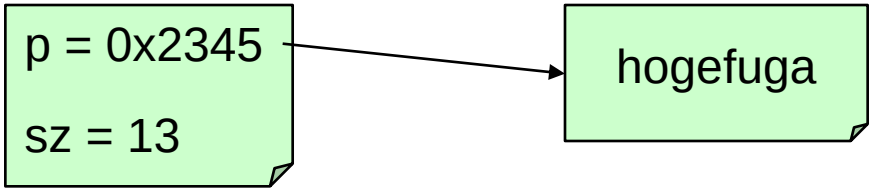
ムーヴで十分！



('h + w' で作られた一時オブジェクト)

str



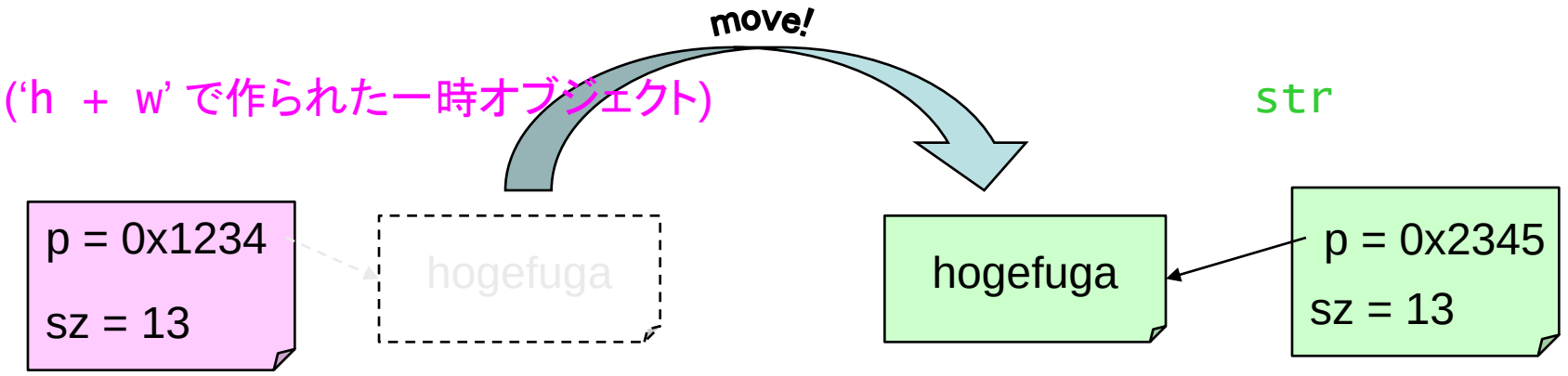



# 一時オブジェクトとコピー

```
String h("Hello, ");
String w("world!");
String str = h + w;
```

'h + w' で作られる一時オブジェクトは
'str' の初期化以外に使われない

ムーヴで十分！



# セッションの流れ

- C++03 におけるコピーの重要性とその限界
- ムーヴの導入 – 動的配列再配置の例から
- ムーヴの活用例
  - ローカルオブジェクトの return
  - 一時オブジェクト
- ムーヴのための C++0x 言語機能 – 右辺値参照
- 明示的なムーヴ
- まとめ

# ムーヴのための C++0x 言語機能

```
String h(“Hello, ”);
String str = h;
```

```
String h(“Hello, ”);
String w(“world!”);
String str = h + w;
```

h は名前の付いたオブジェクト
（左辺値）

‘h + w’ は名前の付いていない
オブジェクト (右辺値) を生成

勝手に壊さないでほしい

‘str’ のコンストラクタは
右辺値を勝手に破壊して O.K.

従来のコピーコンストラクタを起動

ムーヴを用いたコンストラクタを起動

**左辺値と右辺値を自動で区別したい**

# ムーヴのための C++0x 言語機能 – 右辺値参照

従来の参照型 &

＋

新しい参照型 && (右辺値参照型)

2つの参照型でそれぞれ関数を
オーバーロード可能

```
class String {
   .....
   String(const String& x);
   .....
   String(String&& x);
   .....
};
```

従来と同じコピーコンストラクタ

右辺値参照型でオーバーロードした
コンストラクタ (ムーヴコンストラクタ)

# ムーヴコンストラクタの実装 – 文字列クラスの例

```
class String {
  String(const String& x)
    : p_(), sz_()
  { p_ = new char[x.sz_];
    sz_ = x.sz_;
    std::copy(x.p_, x.p_ + x.sz_, p_); }

  String(String&& x)
    : p_(x.p_), sz_(x.sz_)
  { x.p_ = 0;
    x.sz_ = 0; }
  .....
private:
  char *p_; std::size_t sz_;
};
```

コピーコンストラクタ

ムーヴコンストラクタ

# ムーヴコンストラクタの実装 – 文字列クラスの例

```
String h("Hello, ");
String str = h;
```

```
String h("Hello, ");
String w("world!");
String str = h + w;
```

コピーコンストラクタを起動

ムーヴコンストラクタを起動

```
class String {
  .....
  String(const String& x);
  .....
  String(String&& x);
  .....
};
```

# セッションの流れ

- C++03 におけるコピーの重要性とその限界
- ムーヴの導入 – 動的配列再配置の例から
- ムーヴの活用例
  - ローカルオブジェクトの return
  - 一時オブジェクト
- ムーヴのための C++0x 言語機能 – 右辺値参照
- 明示的なムーヴ
- まとめ

# オブジェクトを捨てる

```
std::vector<String> vec;
String str(“Hello, world!”);
vec.push_back(str);
// 以降，‘str’は使わない
```

```
String my_name(“Cryolite”);
int my_age = 28;
Person me(my_name, my_age);
// 以降，‘my_name’は使わない
```

std::vector オブジェクトや
Person オブジェクトに
文字列オブジェクトを
載せるためだけの
文字列オブジェクトたち

従来は，このような場合でもコピーを使っていた

vec や me に渡すときはムーヴで十分

# まとめ

- コピーとともに C++ で重要なムーヴについて，具体例とともに紹介
  - 動的配列の再配置
  - 戻り値へのムーヴ
  - 一時オブジェクトからのムーヴ
- ムーヴを扱いやすくするための C++0x 言語仕様 – 右辺値参照 – について紹介

# このプレゼンで話さなかった大切なこと

- ムーヴした後のオブジェクトはどうなるの？
- ムーヴと例外安全
- 完全な転送 (perfect forward)
- *this とムーヴ

# 右辺値参照に関するオンラインドキュメント

- （日本語での右辺値参照の紹介は少ない）
- “A Proposal to Add Move Semantics Support to the C++ Language”
  - “N1377” で検索 (邦訳有)
- http://d.hatena.ne.jp/ntnek/20090210/
  - “ntnek 右辺値参照” で検索 (邦訳)
- “A Brief Introduction to Rvalue References” (英語)